PATENT APPLICATION TRANSMITTAL LETTER

To Assistant Commissioner of Patents

BOX PATENT APPLICATION

Washington, DC 20231

jc563 U.S. PTO

11/01/99

jc564 U.S. PTO
09/431449
11/01/99

Transmitted herewith for filing of the patent application of:

Inventor(s):   Anthony P. GALLUSCIO, Indialantic, FL
William L. HOLT, Coco Beach, FL
Douglas M. DYER, Indialantic, FL
Albert T. MONTROY, Melbourne, FL

for HIGH SPEED INTERPROCESS COMMUNICATION.

are the following:

    X   Specification, including Abstract
    5   Sheets of drawing (3 sets)
    X   Declaration and Power of Attorney (Unexecuted)
    X   Other: 2 postcards

FOR	NO. FILED	NO. EXTRA		RATE	FEE		RATE	FEE
	CLAIMS AS FILED			Small Entity			Other than a Small Entity	
Basic Fee					$ 380			$
Total Claims	-20- 18	0		x $ 9 =	$		x $ 18 =	$
Indep Claims	-3- 3	0		x $ 39 =	$ 0		x $ 78 =	$
multiple dependent claim present No				x $130 =	$ 0		x $260 =	$
If the difference in Col. 1 is less than zero, enter "0" in Col. 2				TOTAL	$380		TOTAL	$

                    assignment recordal fee $
____ Please charge my Deposit Account No. 17-0055 in the amount
     of $ _____.
____ A check in the amount of _____ is enclosed.
____ The Commissioner is hereby authorized to charge payment of the
     following fees associated with this communication or credit any
     overpayment to Deposit Account No. 17-0055. A duplicate of this
     sheet is enclosed.
        ____ Any additional filing fees required under 37 C.F.R. 1.16.
        ____ Any patent application processing fees under 37 C.F.R. 1.17.
     X   No fee enclosed. No fee is authorized.

11/1/99
Date

Robert J. Sacco
Registration No. 35,667

"EXPRESS MAIL" MAILING LABEL NO.: EE444322314US

QBB\149677.

HIGH SPEED INTERPROCESS COMMUNICATION

Inventor(s): Anthony P. Galluscio
William L. Holt
Douglas M. Dyer
Albert T. Montroy

Exigent International, Inc.

Exigent Docket No.: P-044

CROSS REFERENCE TO RELATED APPLICATIONS

(Not Applicable)

STATEMENT REGARDING FEDERALLY SPONSORED RESEARCH OR DEVELOPMENT

5

(Not Applicable)

BACKGROUND OF THE INVENTION

Technical Field

This invention relates to the field of network interprocess communications

and more particularly to a system and method for high speed interprocess

10 communications.

Description of the Related Art

Interprocess communications (IPC) includes both process synchronization and

messaging. Messaging, or message passing, can be accomplished using pipes,

sockets, or message queues. Pipes provide a byte stream mechanism for

15 transferring data between processes. System calls, for example read and write

calls, provide the underlying mechanisms for reading and writing data to a pipe. As

the writing process writes to a pipe, bytes are copied from the sending process into

a shared data page. Subsequently, when the reading process reads, the bytes are

copied out of the shared data page to the destination process. Sending a message

20 using a pipe consumes a minimum of two memory copies moving a total of 2n

bytes, where n is the number of bytes in the message. Sockets provide an

abstraction mechanism for application programs which can simplify access to

communications protocols. Although network communications provide the primary

impetus for sockets, sockets may be used for IPC, as well. Still, the transmission

25 of data between processes using a socket can consume time necessary to perform

the abstraction overhead, two system calls, for example readv() and writev(), and a minimum of two memory copies moving, in total, 2n bytes. A message queue is an IPC mechanism that is typically managed by an operating system kernel. Typically, the implementation of a message queue is hidden.

Traditional IPC mechanisms can provide IPC between two processes using virtual memory in a shared memory space contained within an operating system kernel. The use of virtual memory implies that, although all processes share the same physical memory space, each process can map a region of the shared memory space differently from other processes. Thus, data residing at one address in the shared memory space can differ physically from the data residing at the same address into the same shared memory space as interpreted by the memory map of a different process.

In traditional IPC, a first process can copy n bytes of data from user memory space into a shared memory space in the operating system kernel. Subsequently, using a system call to the operating system kernel, a second process can copy the same n bytes of data from the shared memory space into the user memory space. Therefore, traditional IPC mechanisms require a minimum overhead of 2n byte copies to communicate n bytes of data between the two processes.

In addition, all methods of traditional IPC require some type of interaction with the operating system kernel. In particular, traditional IPC mechanisms require a minimum of two system calls to the operating system kernel. Moving data in and out of an operating system kernel can include some risk. Specifically, not only must each process move n bytes of data, but each process risks losing CPU control upon invoking the system call required to read or write the data, respectively.

Analogously, computer scientists have recognized the unnecessary expense of passing a message from one process to another. In fact, legacy third-generation programming languages which provide dynamic memory allocation, for example Fortran, inefficiently pass data between processes by copying the data stored in

one region of memory, and storing the data in a different region of memory. Subsequently, the recipient function can process the data before returning a copy of the same using the same mechanism. Recognizing the inefficiencies of this type of message passing, computer scientists have adopted pointer passing as an alternative to data passing when messaging a process. In pointer passing, a recipient process receives only an address of a location in memory of the message data. Subsequently, the recipient can manipulate the data at the passed address, in place, without the need for excessive data copies.

Still, in third-generation languages which have adopted pointer passing, for example C or C + +, the communicating processes ultimately share one memory mapping of a shared memory space for passing data. In fact, in the absence of a single memory map of shared memory space, present methods of pointer passing become unworkeable because data residing at an address in one memory space is not equivalent to the data residing at the same address in another memory space. Therefore, traditional pointer passing cannot be used to resolve the inefficiencies of traditional IPC in which different processes have different memory maps of a shared region of user memory by virtue of the virtual memory scheme associated with network IPC.

In view of the inefficiencies of traditional IPC, traditional mechanisms for IPC are not suitable for real time command and control systems which can require fail-safe and extremely fast conveyancing of information between processes. For example, copying data can be expensive in terms of processor overhead and time delay. In addition, moving data in and out of an operating system kernel can include some risk. Thus, present IPC mechanisms do not provide the level of service required for real-time applications.

SUMMARY OF THE INVENTION

In a preferred embodiment, a method for high speed interprocess communications can comprise four steps. Initially, first and second processes can be attached to a message buffer in a shared region of user memory (RAM).

5   Moreover, message lists corresponding to each of the processes can be established in the shared region. In particular, the attaching step can comprise the steps of: detecting a previously created shared region of user RAM; if a shared region of RAM is not detected, creating and configuring a shared region of user memory for storing accumulated data; and, attaching to the created and configured shared

10 region of RAM. In a preferred embodiment, the attaching step comprises the step of attaching the first and second processes to a message buffer in a shared region of RAM exclusive of operating system kernel space. Message data from the first process can be accumulated in a location in the message buffer.

Advantageously, the message list can be implemented as a message queue

15 using the common data structure, "queue". As a result, subsequent to the accumulating step, a memory offset corresponding to the location in the message buffer can be added to the message queue of the second process. The adding step can comprise the steps of: retrieiving a memory offset in the message buffer corresponding to the location of message data accumulated by the first process;

20 and, inserting the memory offset in the message queue corresponding to the second process. Moreover, the inserting step can comprise the step of atomically assigning the memory offset to an integer location in the message queue corresponding to the second process.

Finally, the accumulated message data at the location corresponding to the

25 memory offset can be processed in the second process. The processing step can comprise the steps of: identifying a memory offset in the message list corresponding to the second process; processing in the second process message data stored at a location in the message buffer corresponding to the memory offset;

and, releasing the message buffer. Consequently, the accumulated message data is transferred from the first process to the second process with minimal data transfer overhead.

Viewed from a system architecture standpoint, a method for configuring high speed interprocess communications between first and second processes can include several steps. Initially, the method can include disposing a message buffer in a shared region of RAM shared between first and second processes. In particular, the disposing step can comprise the steps of: creating and configuring a message buffer in a shared region of RAM exclusive of operating system kernel space; and, creating a message list in the shared region for each process, whereby the message list can store memory offsets of message data stored in the message buffer.

The inventive method can include the step of accumulating message data from the first process in a location in the message buffer and adding the memory offset to a message list corresponding to the second process. Advantageously, the message list can be implemented as a message queue. In consequence, the adding step can comprise the steps of: retrieving a memory offset in the message buffer, the memory offset corresponding to the location of message data accumulated by the first process; and, inserting the memory offset in the message queue corresponding to the second process. Moreover, the inserting step can comprise the step of atomically assigning the memory offset to an integer location in the message queue corresponding to the second process.

Finally, the method can include processing in the second process the accumulated message data stored in the message buffer at a location corresponding to the memory offset. In particular, the processing step comprises the steps of: identifying a memory offset in the message list corresponding to the second process; using in the second process accumulated message data at a location in the message buffer corresponding to the memory offset; and, releasing the message

buffer. Thus, as a result of the inventive method, the accumulated message data can be transferred from the first process to the second process with minimal data transfer overhead.

BRIEF DESCRIPTION OF THE DRAWINGS

There are presently shown in the drawings embodiments which are presently preferred, it being understood, however, that the invention is not limited to the precise arrangements and instrumentalities shown.

5   Fig. 1 is a schematic illustration of a traditional IPC architecture.

Fig. 2 is a schematic illustration of a high-speed IPC architecture in accordance with the inventive arrangements.

Fig. 3 is a process diagram showing the passing of data using a high-speed IPC architecture in accordance with the inventive arrangements.

10   Fig. 4 is a schematic representation of a memory offset.

Fig. 5 is a flow chart illustrating an algorithm for high-speed IPC.

DETAILED DESCRIPTION OF THE INVENTION

The traditional interprocess communication (IPC) architecture includes drawbacks which preclude the use of traditional IPC mechanisms in real time command and control systems which can require fail-safe and extremely fast

5   conveyancing of information between processes. Figure 1 illustrates the commonality between the three traditional mechanisms for IPC. As shown in the figure, two processes 1, 2 communicate using a shared memory space 5 contained within an operating system kernel 4. In particular, process 1 can copy n bytes of data 6 from a user memory space 3 into a shared memory space 5 in the operating

10   system kernel 4. Subsequently, using a system call, process 2 can copy the same n bytes of data 6 from the shared memory space 5 in the operating system kernel 4 into user memory space 3. Therefore, Figure 1 shows a minimum overhead of two system calls and 2n byte copies to communicate n bytes of data between two processes.

15   In contrast, a method for high speed IPC can provide extremely fast IPC both by communicating message data in a shared region of random access memory (RAM) external to the operating system kernel and by limiting the movement of the data. Processes are notified of the location of the message data rather than actually receiving a copy of the message data. The recipient process subsequently

20   can read or manipulate (process) the message data in place. As a result, the number of data copies necessary for high speed IPC is minimized. Notably, a method for high speed IPC utilizes a message list in a message buffer for storing a memory offsets set by atomic assignment. Each memory offset can denote a location in the shared region of RAM where a process attached to the shared region

25   can manipulate data stored therein. Specifically, the message list can be implemented using the common data structure, "queue". When one process messages another, the process need only insert a memory offset to the message data in the recipient's message queue.

Figure 2 provides a high-level perspective of the relationships between the message data 10, the operating system kernel 14, and two processes 11, 12 using high speed IPC. From Figure 2, it will be apparent to one skilled in the art that the message data 10 resides in a shared region of RAM 15 common to both processes 11, 12. Still, one skilled in the art will further recognize that although the shared region of RAM is common to both processes 11, 12, each process 11, 12 can maintain a virtual memory page therein. That is, each process 11, 12 can maintain a different and distinct memory map of the shared region of user RAM 15. However, unlike prior art network IPC where processes do not reconcile differing memory maps of shared RAM, in the preferred embodiment, both processes 11, 12 can reconcile each other's memory mapping into the shared region of RAM 15 by communicating to one another the location of data in the shared RAM relative to a commonly known address.

Capitalizing on this reconciliation, high speed IPC permits the use of the shared region of RAM 15 despite differing memory maps among the processes 11, 12. As a result, in the preferred embodiment, message passing does not require storing message data 10 in operating system kernel space 14. Therefore, in the preferred embodiment, system calls are not required to write and read the data 10. Thus, the elevated risk associated with utilizing operating system kernel space 14 is eliminated. Specifically, the inventive method avoids the risk of a process losing CPU control upon invoking the system call required to read or write the data 10. Hence, the inventive IPC mechanism can provide the level of service required for real-time applications.

Figure 3 illustrates an exemplary conveyance of data 30 between two processes 21, 22 using the inventive method for high speed IPC. Notwithstanding, the invention is not limited in this regard. Rather, the invention can include more than two processes communicating through a shared message store. The conveyance consists of four essential steps. Initially, a first process 22 and a

second process 23 can attach to a small message buffer 25 from a configured pool of message buffers 24 in a shared region of RAM. In the preferred embodiment, the pool of messages buffers 24 can include small 25, medium 26 and large buffers 27. However, the invention is not limited in this regard. Rather, any number or

5   type of message buffers will suffice for operation of the present invention. For instance, a "first-fit" allocation strategy, a "best-fit" strategy, or an "approximate-fit" strategy can also suffice. Still, the preferred combination of all buffers and the management thereof can optimize memory utilization while reducing the cost of memory management.

10   The first process 22 can accumulate message data 30 in a location in the small message buffer 25. Subsequently, the first process 22 can notify a second process 21 of the location of the data 30 in the small message buffer by adding the location of the data 30 into a message list. Specifically, the first process 22 can insert a memory offset 29 of the message data 30 into a message queue 23

15   associated with the second process 21. As shown in Figure 4, a memory offset B represents the number of bytes C from the beginning A of a buffer D, in which data E can be located. In consequence of using memory offsets, rather than absolute addresses (pointers), two processes can reference a single piece of data in a common region of RAM, despite having different memory maps of the memory

20   region. Hence, although the first process 22 and the second process 21 may have differing memory maps of the small message buffer 25, the memory offset 29 can indicate to each process 21, 22 the number of bytes from a common address of the small message buffer 25 in which the message data 30 can be located.

      Message queues 23, 28 preferably are created in the shared region of RAM.

25   Each message queue 23, 28 is a list of messages which can be represented by the common data structure, "queue", which, in the preferred embodiment, can handle integer values in a first-in-first-out (FIFO) order. Each message queue 23, 28, alternatively referred to as an "inbox", can contain an administrative area having

variables for administering the queue of integer offsets. Those variables may include variables for tracking the position of the front and rear elements of the queue and the queue size.

The first process 22 can access the message queue 23 of the second process 21 by addressing the message queue 23 by name. The first process 22 can either have a priori knowledge of the name of the message queue 23, or the first process 22 can rely on a naming service. Specifically, the first process 22 can cross-reference in a naming service the process identification number corresponding to the second process 21 with the location of the message queue 23 of the second process 21. The naming service can be as simple as a file that contains names of message queues mapped to processes. The naming of message queues can depend on the nature of the specific operating system. For instance, in the Windows NT operating system, the operating system names the message queue. In contrast, the Unix operating system uses integer identifiers to identify a message queue.

The memory offset 29 can be logically inserted in the message queue 23 of the second process 21, but advantageously, because the memory offset 29 can be internally represented as an integer, the memory offset 29 can be physically assigned to a data member in a node in the message queue 23 using a simple integer assignment available, for instance, in the C, C++ or Java programming languages. The mechanism for assignment can vary depending on the implementation of the queue data structure. However, as an example, the first process 22 can calculate the address of the first element in the message queue 23, and can make an atomic assignment of the memory offset 29 to that address.

Specifically, in C-syntax, the physical assignment can consist of "*(inbox_address + front_of_queue) = offset_of_message_data ". This C-style statement can atomically assign the memory offset 29 of the message data 30 to the address of the first element in message queue 23. The assignment can be

atomic in that a single instruction is required, e.g. "newValue = 5". In the case of an atomic assignment, the entire integer memory offset 29 can be written to the message queue 23 using the single instruction. The atomic assignment can be contrasted with the case of copying a data message using a memory copy, such as "memcpy", which performs an assignment for each byte in the data message.

The front_of_queue variable can be stored at a pre-determined location, as specified by the message queue structure in the beginning of the small message buffer 25. Still, one skilled in the art will recognize that the message queue 23 needn't be stored in the small message buffer 25. Rather, the message queue 23 can be stored in another message buffer, into which access can be provided using any of the traditional IPC techniques. Alternatively and advantageously, access to the message buffer could occur using high speed IPC.

Finally, the second process 21 can identify the memory offset 29 placed in the corresponding message queue 23. Specifically, the second process 21 can poll the message queue 23 waiting for a new memory offset 29 to arrive. Alternatively, the first process 22 can signal the second process 21 that new message data 30 has arrived. Either mechanism can be acceptable depending upon specific application requirements.

Having identified the memory offset 29, the second process 21 can manipulate the accumulated message data 30 in place in the small message buffer 25 corresponding to the memory location denoted by the memory offset 29. The second process 21 can use the accumulated data 30 in accordance with the unique data requirements of the second process 21. When finished, the second process 21 can release the small message buffer 25 using conventional memory management techniques.

Figure 5 is a flow chart describing a method for high speed IPC. The flow chart depicts a single process which can communicate with another process using the inventive method. As shown in the drawings, the method begins in step 100

where a process can attempt to attach to a message buffer in a shared region of RAM, exclusive of the operating system kernel. One skilled in the art will recognize that each attempt to attach to the message buffer will include moderation by a locking mechanism in order to prevent a logic race condition. One such example of a locking mechanism is a mutex which allows an atomic check and set of a variable that protects a shared region. If one process has the mutex, other processes are blocked from accessing the shared region until the mutex is released.

In decision step 102, if the process is the first activated process in the system, in step 104, a message buffer in the shared region of RAM is created and, in step 106, configured. Preferably, the process creating the shared region of RAM obtains a mutex and releases the mutex only when the shared region is created and configured. The release of the mutex acts as notification to other interested processes that the shared region of RAM is ready for use.

Configuring the shared region of RAM can include naming the shared region, initializing the shared region variables in an administrative area, and sizing the buffer pools. Notably, the shared region may be configured using a stored configuration that is merely retrieved by the process and applied to the shared region.

Whether the process creates and configures a new shared region or attaches to a previously created shared region, in step 108, the process can create a message queue in the shared region corresponding to the process. In particular, the message queue can be used to store incoming memory offsets, placed in the message queue by other processes. Having attached to a message buffer and created a message queue, in step 110, the process can perform normal intraprocess operations until a need for IPC arises, either where the process is a recipient or sender of a message, as determined in decision step 112.

If the process is a first process attempting to transmit data to a second process, the first process, in step 122 can obtain a memory offset to free memory

13

space in the message buffer. One skilled in the art will recognize that obtaining a memory offset to free memory requires the use of a memory management mechanism for allocating buffers in a shared region of user memory. Still, one skilled in the art will futher recognize the widespread availability of memory management mechanisms suitable for accomplishing the same. For example, just as the "malloc()" function included as part of the ANSI C standard library can abstract the details of memory management, a buffer pool allocator for high speed IPC can abstract the details of managing memory offsets into the message buffer. The buffer pool allocator for high speed IPC can be implemented using techniques well known in the art and thoroughly documented in Kernighan and Ritchie, "The C Programming Language: 2nd Edition", pp. 185-189, incorporated herein by reference.

Subsequently, in step 124, the first process can accumulate message data for the benefit of the second process with the writing beginning at the location corresponding to the memory offset. When finished accumulating the message data in the message buffer, in steps 126 and 128, the first process can place the memory offset in the message queue corresponding to the second process. Significantly, the placement of the memory offset can be an atomic assignment to an integer location in the shared region of RAM. The act of placing the memory offset in the message queue is tantamount to notifying the second process of an attempt at IPC.

Correspondingly, if the process is a second process receiving a request for IPC from a first process, in step 114, the second process can identify a memory offset in the message queue corresponding to the second process. In step 116, the second process can retrieve the memory offset, and in step 118, the second process can use the memory offset to access the data accumulated by the first process at an appropriate location in the message buffer. Significantly, because the accumulated data is stored in a shared region of user memory, it is not necessary

for the second process to copy the accumulated data to a different memory space. Rather, in step 120, when finished using the data, the second process need only release the buffer using the above-identified buffer pool allocator.

As illustrated in Figure 4, the significant differences between the inventive method and traditional IPC mechanisms include the present method's use of a shared region of RAM to store accumulated data. As a result of the use of the shared region, the inventive method does not require operating system calls to write and read accumulated data. In addition, because the present method uses a shared region of RAM instead of a memory region in the operating system kernel, the reconfiguration of the shared region does not require the rebooting of the operating system. Finally, high speed IPC provides a faster and safer mechanism for IPC in that the overhead associated with IPC is minimized from two system calls and 2n bytes of data movement to a minimal n bytes of data movement.

CLAIMS

1. 1. A method for high speed interprocess communications comprising the steps
2. of:
3. attaching first and second processes to a message buffer in a shared region
4. of random access memory (RAM) exclusive of operating system kernel space, each
5. said process having a message list;
6. accumulating message data from said first process in a location in said
7. message buffer;
8. adding to said message list of said second process a memory offset
9. corresponding to said location in said message buffer; and,
10. processing in said second process said accumulated data at said location
11. corresponding to said offset,
12. whereby said accumulated message data is transferred from said first
13. process to said second process with minimal data transfer overhead.

1. 2. The method according to claim 1, wherein the attaching step comprises the
2. steps of:
3. detecting a previously created shared region of RAM;
4. if a shared region of RAM is not detected, creating and configuring a shared
5. region of RAM for storing accumulated data; and,
6. attaching said first and second processes to said shared region.

1. 3. The method according to claim 1, wherein said message list is a message
2. queue.

1. 4. The method according to claim 3, wherein the adding step comprises the
2. steps of:
3. retrieving a memory offset in said message buffer corresponding to said

4   location of data accumulated by said first process; and,

5   inserting said memory offset in said message queue corresponding to said

6   second process.

1   5.   The method according to claim 4, wherein the inserting step comprises the

2   step of atomically assigning said memory offset to an integer location in said

3   message queue corresponding to said second process.

1   6.   The method according to claim 1, wherein the processing step comprises the

2   steps of:

3   identifying a memory offset in said message list corresponding to said second

4   process;

5   processing in said second process message data stored at a location in said

6   message buffer corresponding to said memory offset; and,

7   releasing said message buffer.

1   7.   A method for configuring high speed interprocess communications between

    first and second processes comprising the steps of:

        disposing a message buffer in a shared region of random access memory

    (RAM) shared between said first and second processes;

5   accumulating message data from said first process in a location in said

6   message buffer;

7   adding to a message list corresponding to said second process a memory

8   offset corresponding to said location in said message buffer; and,

9   processing in said second process said accumulated message data stored in

10   said message buffer at a location corresponding to said offset,

11   whereby said accumulated message data is transferred from said first

12   process to said second process with minimal data transfer overhead.

13   8.   The method according to claim 7, wherein the disposing step comprises the

14   steps of:

15   creating and configuring a message buffer in a shared region of RAM

16   exclusive of operating system kernel space; and,

17   creating a message list in said shared region for each said process,

18   whereby said message list can store memory offsets of message data stored

19   in said message buffer.

1   9.   The method according to claim 7, wherein said message list is a message

2   queue.

10.   The method according to claim 9, wherein the adding step comprises the

steps of:

        retrieving a memory offset in said message buffer, said memory offset

corresponding to said location of said message data accumulated by said first

process; and,

        inserting said memory offset in said message queue corresponding to said

second process.

11.   The method according to claim 10, wherein the inserting step comprises the

2   step of atomically assigning said memory offset to an integer location in said

3   message queue corresponding to said second process.

1   12.   The method according to claim 7, wherein the processing step comprises the

2   steps of:

3   identifying a memory offset in said message list corresponding to said second

4   process;

5   processing in said second process said accumulated message data at a

6   location in said message buffer corresponding to said memory offset; and,

7   releasing said message buffer.

1   13.   A computer apparatus programmed with a set of instructions stored in a

2   fixed medium for high speed interprocess communications, said programmed

3   computer apparatus comprising:

4   means for attaching first and second processes to a message buffer in a

5   shared region of random access memory (RAM) exclusive of operating system

6   kernel space, each said process having a message list;

7   means for accumulating message data from said first process in a location in

8   said message buffer;

9   means for adding to said message list of said second process a memory

10   offset corresponding to said location in said message buffer; and,

11   means for processing in said second process said accumulated data at said

12   location corresponding to said offset.

1   14.   The computer apparatus of claim 13, wherein the attaching means

2   comprises:

3   means for detecting a previously created shared region of RAM; and,

4   means for creating and configuring a shared region in RAM for storing

5   accumulated data if a previously created shared region of RAM is not detected by

6   said detecting means.

1   15.   The computer apparatus according to claim 13, wherein said message list is

2   a message queue.

1   16.   The computer apparatus according to claim 15, wherein the adding means

2   comprises:

3   means for retrieving a memory offset in said message buffer corresponding

4   to said location of data accumulated by said first process; and,

5   means for inserting said memory offset in said message queue corresponding

6   to said second process.

1   17.   The computer apparatus according to claim 16, wherein the inserting means

2   comprises means for atomically assigning said memory offset to an integer location

3   in said message queue corresponding to said second process.

1   18.   The computer apparatus according to claim 13, wherein the processing

2   means comprises:

3   means for identifying a memory offset in said message list corresponding to

4   said second process;

5   means for using in said second process message data at a location in said

6   message buffer corresponding to said memory offset; and,

7   means for releasing said message buffer.

ABSTRACT

A method for high speed interprocess communications comprises four steps. Initially, first and second processes can be attached to a message buffer in a shared region of user memory. In addition, each process can have a corresponding message queue. In a preferred embodiment, the attaching step comprises the step of attaching first and second processes to a message buffer in a shared region of user memory exclusive of operating system kernel space. Second, message data from the first process can be accumulated in a location in the message buffer. Third, a memory offset corresponding to the location in the message buffer can be placed in the message queue of the second process. Finally, the accumulated data at the location corresponding to the offset can be used in the second process. Consequently, the accumulated message data is transferred from the first process to the second process with minimal data transfer overhead.

1

2

3

Process

USER SPACE

Process

n bytes of data

6

6

n bytes of data

6

n bytes of data

OPERATING SYSTEM KERNEL

5

4

FIG. 1

_13_

USER SPACE

_11_

Process

n bytes of data

Process

_12_

_15_

_10_

OPERATING SYSTEM KERNEL

_14_

FIG. 2

21

23

process

offsets

22

process

2

29

29

1

28

25

small buffers

30

data

3

24

medium buffers

26

large buffers

27

FIG. 3

D

A   E

| 0 | 1 | 2 | ... | n-1 | n |

                                   DATA (n)

n bytes

B

C

FIG. 4

Connect to SR

1st Up ?

Create SR

SR Config

Y

N

Create Inbox

Do Work

114   N

Write   R or W ?   Read   Offset in Q

112

Get Buffer Offset

Use Buffer

Put Offset   Receive Q

Y

Get Offset

116

Use Data

118

Release Buffer

126

FIG. 5

DECLARATION FOR PATENT APPLICATION

Docket No: 6572-14

As a below named inventor, I hereby declare that:

My residence, post office address and citizenship are as stated below next to my name.

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled _HIGH SPEED INTERPROCESS COMMUNICATION_
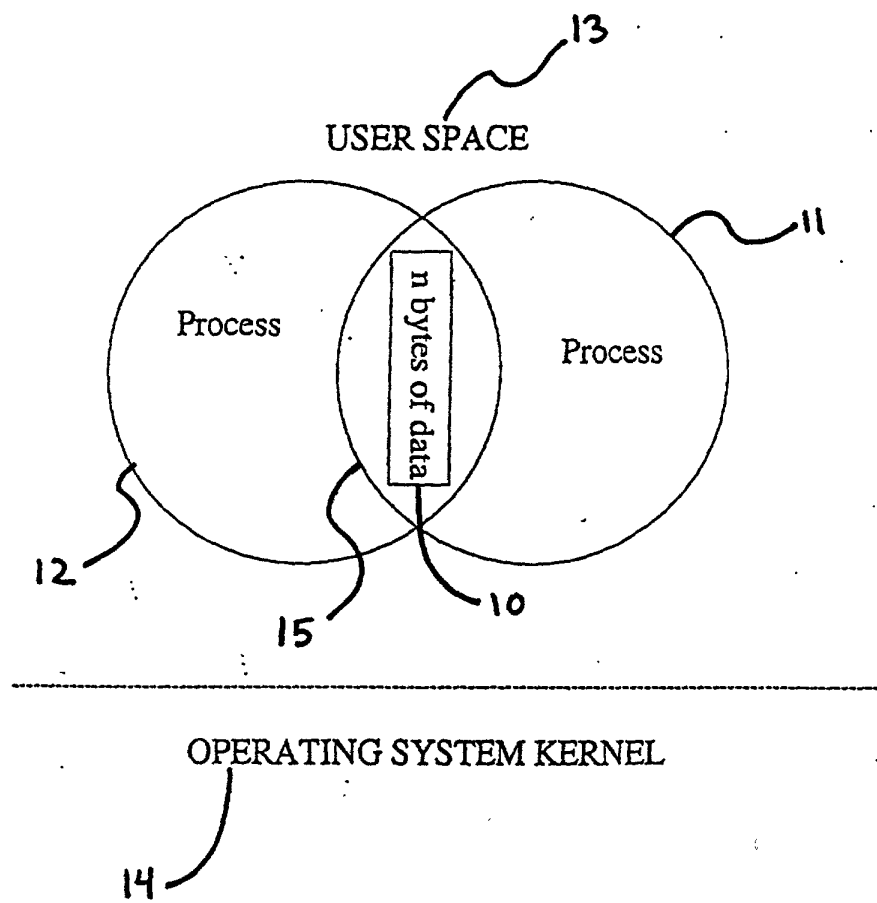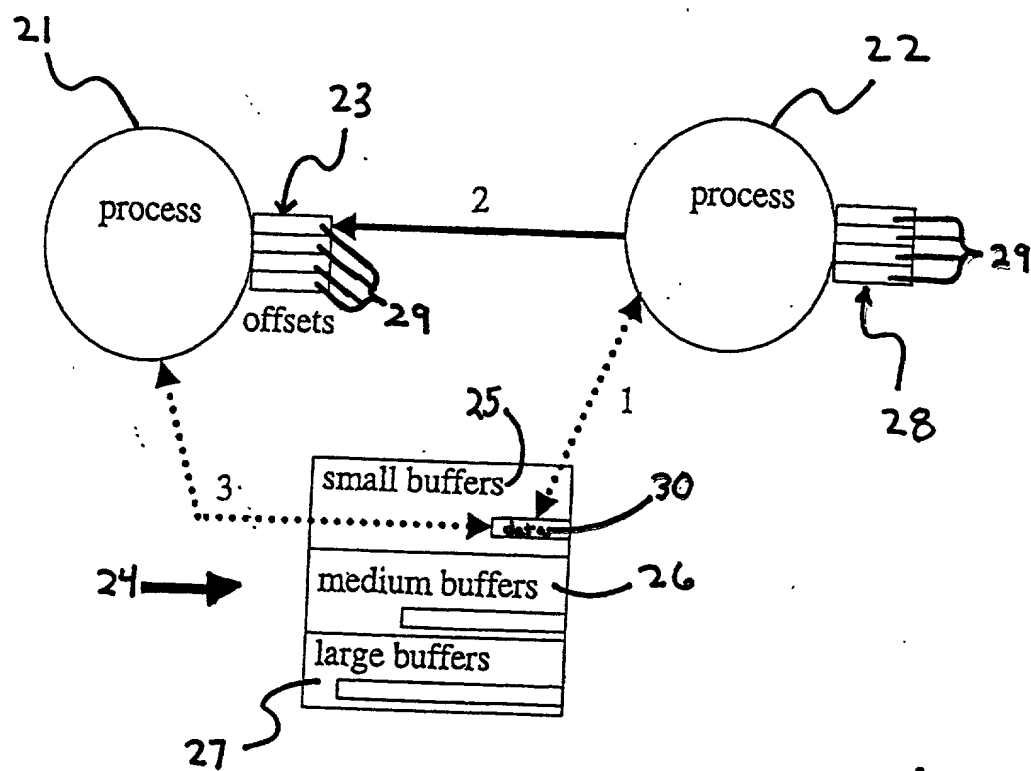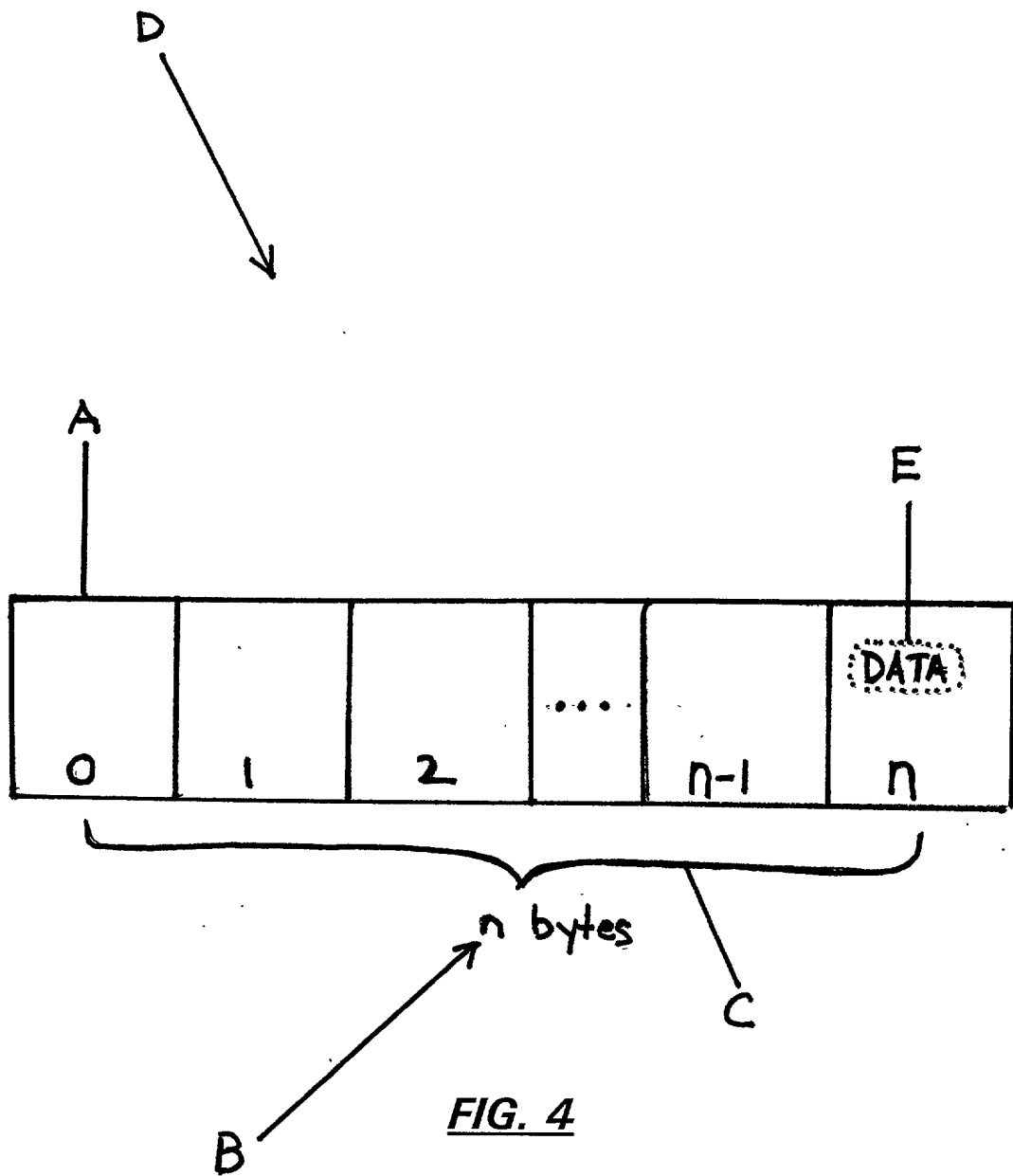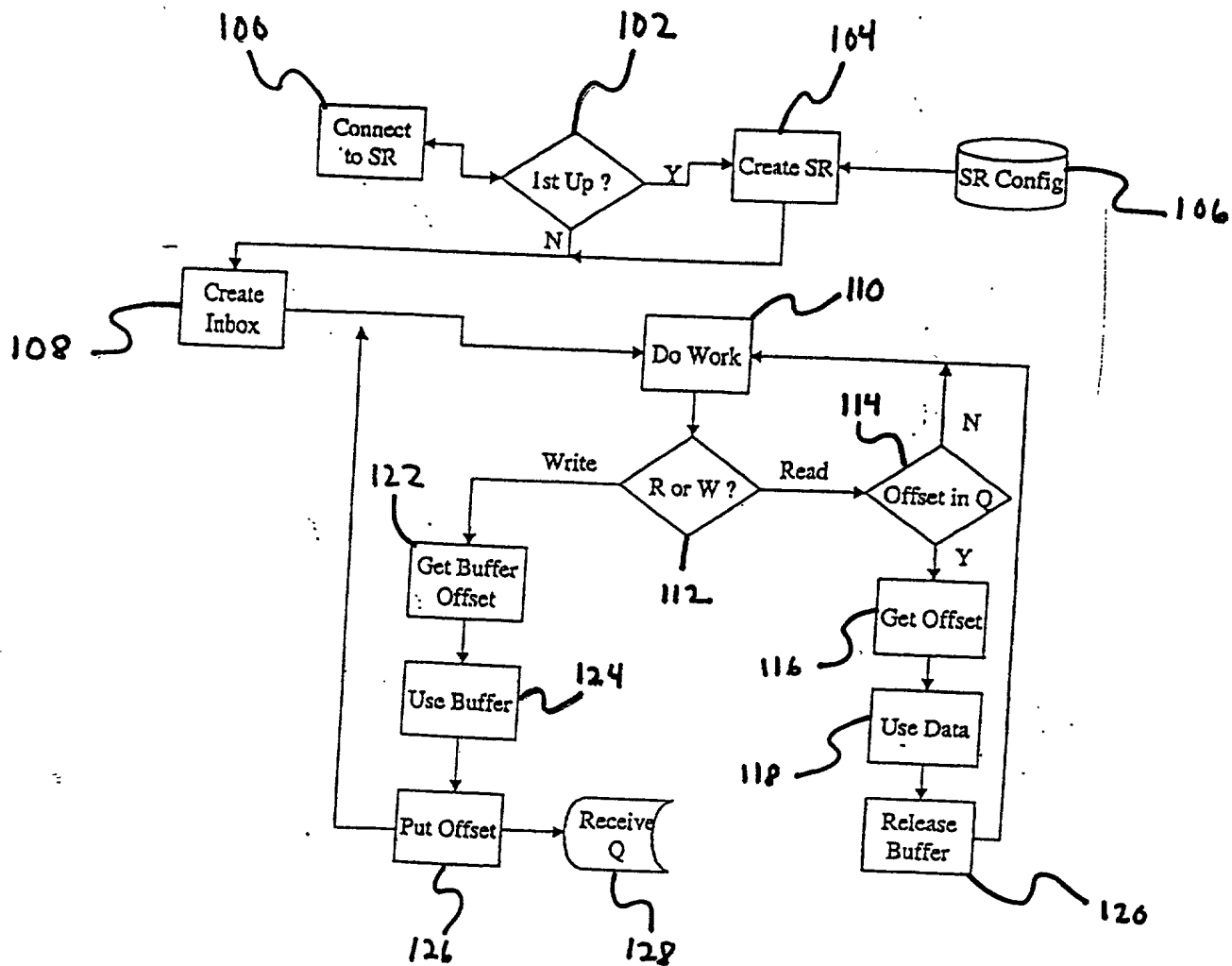
the specification of which (check one)

        _X_   is attached hereto.
       ___   was filed on _____ as
            Application Serial No. _____ and
            was amended on _____ (if applicable).

I hereby state that I have reviewed and understand the contents of the above identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose information which is material to the examination of this application in accordance with Title 37, Code of Federal Regulations Section 1.56(a).

I hereby claim foreign priority benefits under Title 35, United States Code 119 of any foreign application(s) for patent or inventor's certificate listed below and have also identified below any foreign application for patent or inventor's certificate having a filing date before that of the application on which priority is claimed:

Prior Foreign Application(s)   Priority Claimed

                                                    Yes No

_____ _____ _____

  (Number)   (Country)   (Day/Month/Year Filed)

I hereby claim the benefit under Title 35, United States Code, Section 120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code, Section 112, I acknowledge the duty to disclose material information as defined in Title 37, Code of Federal Regulations, Section 1.56(a) which occurred between the filing date of the prior application and the national or PCT international filing date of this application:

_____ _____ _____

(Appln. Serial No.)   (Filing Date)   (Status-patent, pending,
                                                 abandoned)

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

I hereby appoint J. Rodman Steele, Jr., Registration No. 25,931; Gregory A. Nelson, Registration No. 30,577; Joseph W. Bain, Registration No. 34,290; Robert J. Sacco, Registration No. 35,667; Mark D. Passler, Registration No. 40,764; Stanley A. Kim, Registration No. 42,730; Steven M. Greenberg, Registration No. 44,725; and Scott D. Paul, Registration No. 42,984 as my attorneys with full power of substitution and revocation to prosecute this application and to transact all business in the Patent and Trademark Office connected therewith, to amend the specification, to appeal in case of rejection, as they may deem advisable, to receive the patent when granted and generally to do all matters and things needful in the premises as fully and to all intents and purposes as I could do.

Please direct all correspondence to: _____ROBERT J. SACCO_____

QUARLES & BRADY LLP
222 Lakeview Avenue - Suite 400
P. O. Box 3188
West Palm Beach, Florida 33402-3188
Telephone: (561) 653-5000

Full name of sole or first inventor __Anthony P. Galluscio__

Inventor's signature _____ Date _____

Residence __Indialantic, Florida__

Citizenship _____USA__

Post Office Address __549 Humming Bird Drive, Indialantic, FL 32903__

Full name of second joint inventor, if any __William L. Holt__

Second Inventor's signature _____ Date _____

Residence __Coco Beach, Florida__

Citizenship _____USA__

Post Office Address __57 S. Atlantic Avenue, Coco Beach, Florida 32931__

Full name of third joint inventor, if any __Douglas M. DYER__

Second Inventor's signature _____ Date _____

Residence __Indialantic, Florida__

Citizenship _____USA__

Post Office Address __135 Eighth Avenue, Indialantic, Florida 32903__

Full name of fourth joint inventor, if any ___Albert T. MONTROY_____

Second Inventor's signature _____ Date _____

Residence ____Melbourne, Florida_____

Citizenship _____USA_____

Post Office Address ___4260 Windover Way, Melbourne, Florida 32934_____